Geolocalització amb Android sense utilitzar els serveis de Google

Autor: Marc Mas i Divins

Data: 12/03/2018

Director: Francisco del Aguila Lopez

Escola Politècnica Superior d'Enginyeria de Manresa
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Índex

Imatges

Taules

1 Introducció

1.1 Formulació del problema

Actualment existeixen moltes aplicacions mòbil per a obtenir i processar les dades de localització, tant fora de línia com en línia. El problema principal és que la majoria d'aquests programes utilitzen els serveis de Google i, per tant, podem perdre el control sobre les nostres dades o fer més difícil l'administració i gestió d'aquestes.

Un altre punt és el fet que s'està expandint la preocupació sobre la protecció de dades dels usuaris a causa de noves notícies que indiquen un ús poc ètic de les dades de part d'algunes empreses i que ha comportat una creixent inquietud sobre la confiança en aquestes. És normal pensar que aquesta preocupació pot augmentar quan parlem de la possibilitat que una companyia tingui les dades de la teva ubicació i les localitzacions on has passat i pugui vendre-les o utilitzar-les. Aquest fet encara és més preocupant quan es vol compartir amb amics la teva ubicació actual, ja que és difícil trobar un medi privat, fiable i transparent pel qual compartir les teves dades.

1.2 Objectius del projecte

Un cop definit el problema del projecte podem separar els objectius en quatre:

1. Aportar i donar a conèixer eines que permetin gestionar i processar les dades de localització de forma totalment transparent i sense la necessitat dels serveis de Google, ja sigui en línia com fora de línia.

2. Oferir la possibilitat de compartir les teves dades, mentre es realitza una ruta o un cop finalitzada, per un canal directe sense la intervenció de tercers.

3. Permetre a l'usuari tenir un control total sobre les seves dades.

4. Adquirir coneixement sobre l'obtenció i processament de les dades, i aprendre a fer un ús ètic.

2 Estat de l'art

Certament hi ha moltes aplicacions mòbil de software lliure que permeten gestionar i processar les dades GPS, però no n'hi ha cap que no depengui d'una manera o altra dels serveis de Google, ja sigui per obtenir la ubicació, com per visualitzar-la en un mapa. Altres problemes d'aquestes aplicacions és que fan servir tercers per a comunicar i enviar la localització, que provoca que es perdi el control total sobre les dades, o el fet que no ofereixen cap mètode de compartir les teves dades mentre realitzes una ruta.

D'aquesta manera s'ha trobat diverses aplicacions que compleixen molts dels objectius. Les principals serien els següents:

- **OSMTracker[13]**: Aquesta aplicació de software lliure permet crear rutes utilitzant el mateix GPS del mòbil i visualitzar les rutes utilitzant un proveïdor de mapes lliure. Aquesta aplicació també permet compartir les teves rutes amb diferents opcions de privacitat un cop finalitzades. El problema bàsic d'aquesta aplicació és que no permet compartir en temps real les rutes que es realitzen.

- **Owntracks[14]**: Aquesta aplicació de software lliure permet crear rutes i compartir-les amb els amics per un canal segur de manera que es pot assegurar que les dades es mantenen privades. Tot i això aquesta aplicació fa ús dels serveis de Google per a obtenir la ubicació i per a visualitzar-la.

- **Hypertrack[8]**: Una altra aplicació de software lliure que permet crear rutes i compartir-les, tot i que aquesta aplicació no utilitza Google per obtenir la ubicació sí que l'utilitza per visualitzar-la.

Com aquestes aplicacions n'hi ha altres, però aquestes són les que compleixen més requisits.

3 Definició de l'abast

3.1 Abast

Per a assolir els objectius proposats s'ha decidit a desenvolupar una aplicació mòbil que permeti l'adquisició, processament, visualització i enviament de dades GPS. Un servei web on es pugui rebre, processar i visualitzar les dades GPS enviades per l'aplicació. Per assolir-ho, primer de tot, s'hauran de fixar uns passos a seguir.

Inicialment s'haurà de realitzar un estudi dels diferents mètodes d'obtenció de dades GPS sense la necessitat de Google. Seguidament es realitzarà la cerca de diferents alternatives per mostrar les dades obtingudes. Per a poder compartir les dades i guardar-les s'exploraran diferents opcions de desenvolupament web. Un cop analitzat els diferents mètodes se seleccionarà el que més s'adeqüi a les necessitats del projecte.

Un cop seleccionats els mètodes es procedirà a realitzar un calendari amb fites a assolir per assegurar que el projecte es desenvolupa correctament. Finalment es procedirà a realitzar l'elaboració del codi del projecte.

3.2 Possibles obstacles

Durant la realització de tot projecte sempre hi ha obstacles i imprevistos que poden provocar un retràs en la seva realització o fins i tot portar al fracàs. Per reduir el màxim el risc analitzem els possibles obstacles.

3.2.1 Utilitzar eines poc madures

Per portar a terme aquest projecte s'utilitzaran eines de software lliure per evitar la necessitat d'usar els serveis de Google. El problema apareix, quan pel fet que la majoria d'usuaris utilitzen Google, les eines que es poden trobar pot ser que ja no estiguin suportades pels

desenvolupadors, poden ser poc madures i no oferir els suficients recursos o poden tenir un gran risc d'abandonament per aquests. Per reduir el màxim aquest risc es farà una busca intensiva de diferents mètodes que es puguin utilitzar per substituir els serveis de Google.

3.2.2 Calendari

El calendari per a realitzar qualsevol projecte és una part essencial i molt important, ja que marcarà el correcte desenvolupament d'aquest si es compleixen les fites marcades. Per tant, per evitar qualsevol imprevist es realitzarà un calendari amb diverses fases amb les seves conseqüents fites per assegurar que si es produís un imprevist, no comportés el fracàs de tot el projecte.

4 Planificació temporal

4.1 Pla de projecte

4.1.1 Estudi de l'abast i cerca d'informació

La fase inicial consistirà en la cerca d'informació per conèixer l'estat de l'art sobre l'àmbit del projecte. En aquesta fase es buscarà els diferents mètodes i alternatives per arribar a la solució del problema i es definirà l'abast del projecte. Finalment es realitzarà un esbós de l'estructura del projecte amb les possibles solucions i es realitzarà una reunió amb el professor per seleccionar la més adient.

4.1.2 Proves de la solució i preparació de l'entorn de treball

Un cop elegida la solució es realitzarà un període de proves per aprendre a utilitzar àgilment les eines seleccionades per portar a terme la solució. També es preparà l'entorn de treball creant un gestor de versions pel projecte i instal·lant els possibles programes necessaris per a la solució escollida.

4.1.3 Inici de la solució

Amb una certa agilitat adquirida amb les noves eines de treball en la fase anterior s'inicia la fase inicial de la solució. En aquesta fase es crearà la versió inicial de l'aplicació mòbil i de l'aplicació web. Es definirà l'estructura de la base de dades i les diferents classes per gestionar les dades que s'obtinguin. També s'acabarà de definir les característiques essencials que volem que tinguin tant l'aplicació web com l'aplicació mòbil.

4.1.4 Definició de la comunicació entre aplicació web i mòbil

Un cop definit en la fase anterior l'estructura de la base de dades i la informació a compartir entre les dues aplicacions es procedirà a definir la comunicació entre web i mòbil.

4.1.5 Desenvolupament del projecte

Un cop definida la solució, l'abast, l'estructura bàsica de la gestió de les dades, tant amb la base de dades, les classes i la comunicació, procedirem a realitzar el projecte i a elaborar el codi conseqüent. Aquesta fase serà la més llarga i ocuparà la major part del projecte. Com es pot suposar aquesta etapa pot estar sotmesa a diversos canvis a causa del contingut que abasteix i el temps.

4.1.6 Documentació

Finalment un cop assolit la solució del projecte es procedirà a realitzar-ne la documentació.

4.2 Duració aproximada

Fase	Duració aproximada (h)
Estudi de l'abast i cerca d'informació	40
Proves de la solució i preparació de l'entorn de treball	25
Inici de la solució	40
Definició de la comunicació entre aplicació web i mòbil	40
Desenvolupament del projecte	340
Finalització i documentació	60
Total	545

Taula 1: Temps aproximat en hores

4.3 Diagrama de Gantt

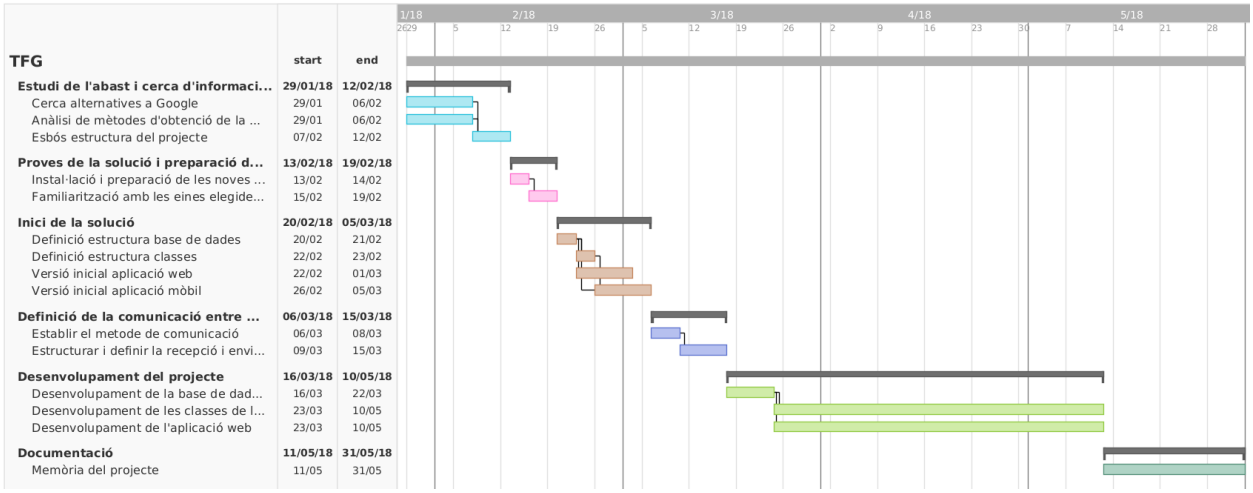


Figura 1: Diagrama de Gantt

5 Elecció de la solució

Per a demostrar que no és necessari dependre dels serveis de Google per a obtenir, processar i visualitzar les dades GPS s'ha decidit realitzar una aplicació mòbil amb les següents opcions:

- **Crear rutes:** Poder emmagatzemar les dades GPS de les nostres rutes i visualitzar-les en un mapa demostrarà que no és necessari els serveis de Google.
- **Compartir rutes:** Poder compartir les rutes en temps real sense intermediaris per provar que també es pot aconseguir sense necessitat de tercers.

Per poder dur a terme la segona opció del mòbil i promoure la idea que els usuaris han de tenir un control absolut sobre les seves dades s'ha decidit crear un servei web. Aquest servei web s'ha decidit que serà semblant a una xarxa social, ja que és en aquestes on les dades es veuen més exposades. La web oferirà la possibilitat d'esborrar les teves rutes i també d'esborrar el propi usuari. En cas que s'esborri l'usuari s'hauran d'esborrar absolutament totes les dades.

5.1 Android Studio

S'ha decidit desenvolupar l'aplicació mòbil en l'entorn de desenvolupament d'Android Studio [2], ja que el sistema operatiu Android és un sistema obert amb una comunitat molt gran que hi dóna suport i, a més a més, és molt més fàcil testejar i compartir els resultats. Una l'alternativa era fer servir Xamarin [18], aquest entorn de desenvolupament permet crear aplicacions simultàniament per Android i iOS, però comprovar el correcte funcionament en els mòbils amb sistema operatiu iOS comporta bastants problemes.

5.2 Web2py

Per a desenvolupar el servei web s'ha decidit utilitzar el framework Web2py[17] enfront de Django[7], ja que teníem experiència en el desenvolupament amb Web2py i d'aquesta manera reduíem el temps en l'aprenentatge en el desenvolupament d'un nou framework.

6 Desenvolupament del sistema

Un cop elegida la solució del problema es procedeix a desenvolupar l'aplicació mòbil i el servei web.

6.1 Aplicació mòbil

6.1.1 Estructura aplicació mòbil

Per desenvolupar l'aplicació mòbil s'ha seguit l'arquitectura model-vista-presentador (MVP)[12], veure 2.

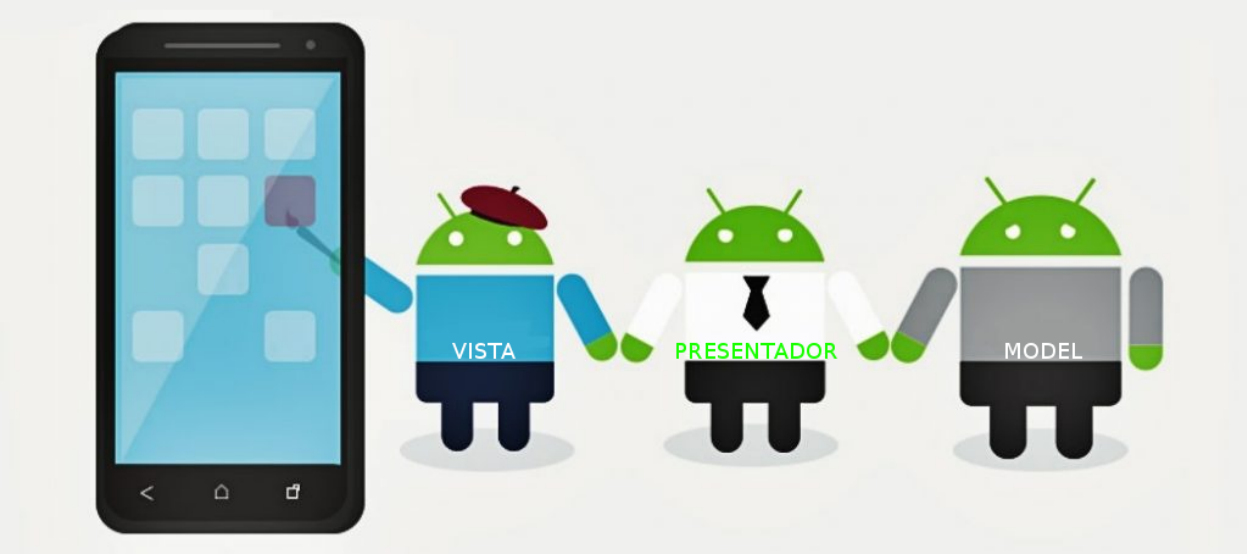


Figura 2: Base de dades de l'aplicació

- **Model:** És una interfície que defineix les dades a visualitzar o d'interfície d'usuari. En aquesta aplicació és en la base de dades on es guardaran totes les dades GPS.
- **Presentador:** Actua sobre el model i la vista Recupera dades de repositoris (el model) i formateja les dades de la vista al visualitzador.
- **Vista:** És una interfície passiva que visualitza les dades (el Model) i encamina comandes d'usuari (events) al Presentador.

Un cop fixada l'arquitectura a seguir hem desenvolupat l'estructura de la nostra aplicació, veure [3].

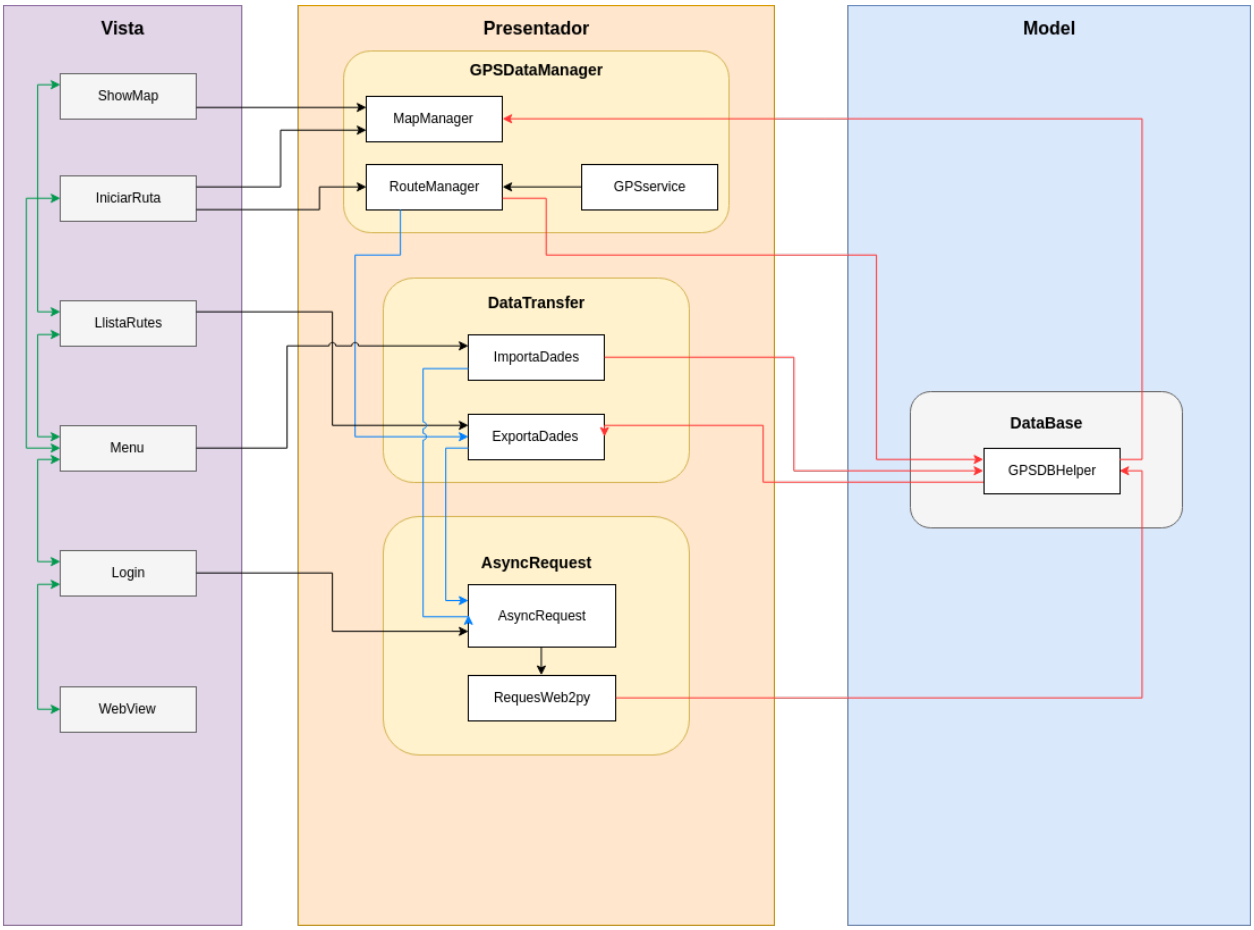


Figura 3: Estructura de l'aplicació

6.1.2 Model

La base de dades del nostre sistema només s'encarrega de guardar les dades GPS. Altres dades com les de l'usuari, les quals se'n voldrà mantenir la persistència, seran guardades utilitzant SharedPreferences que s'explicaran més endavant.

L'estructura de la base de dades del mòbil és la següent:

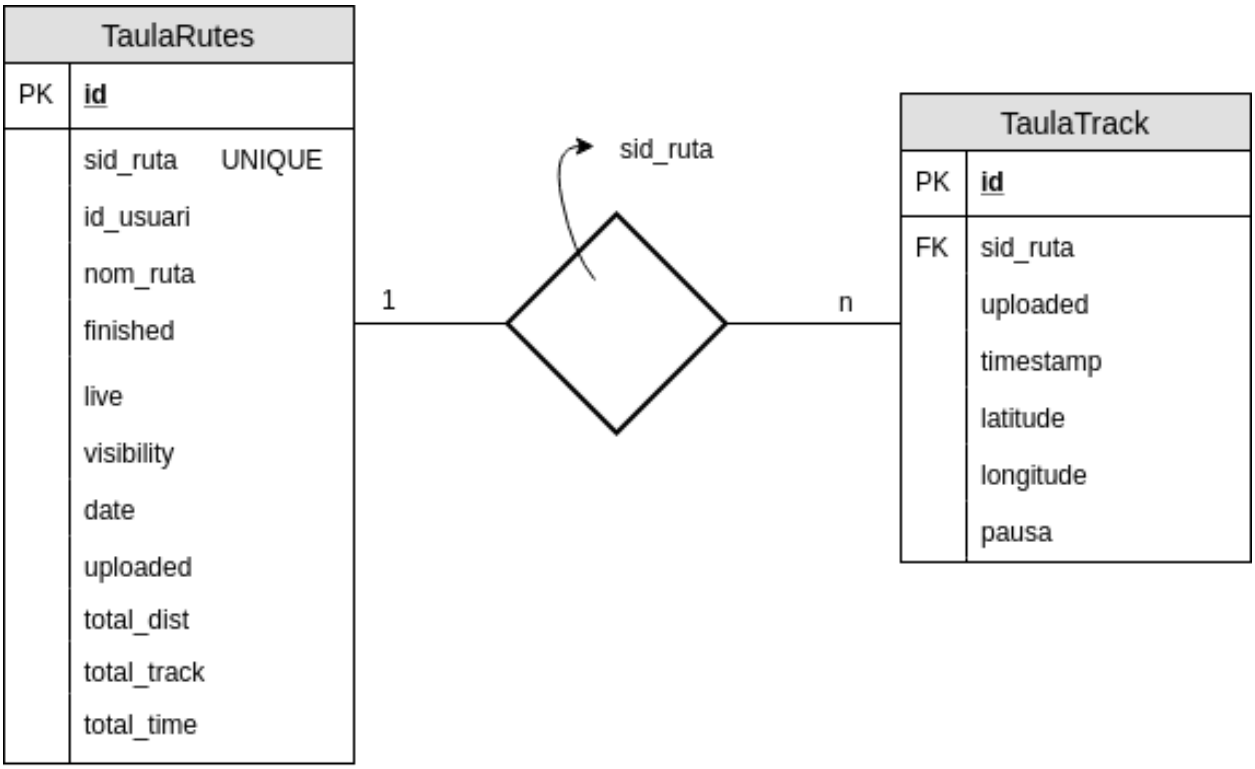


Figura 4: Base de dades de l'aplicació

Com podem veure tenim dues taules, la taula rutes i la taula track. La taula rutes és la que identificarà una ruta en concret i està composta pels següents camps:

- **id [int]:** Identificador únic de la ruta.
- **sid_ruta [text]:** Un conjunt de 20 caràcters compost pel nom de l'usuari i la data de la creació de la ruta. Aquest camp és únic, ja que no és possible que un mateix usuari creï dues o més rutes en el mateix instant de temps. Aquest camp s'utilitza per identificar les rutes entre el servei web i l'aplicació mòbil.

- **id_usuari [int]:** Identifica la *id* de l'usuari del servei web.
- **nom_ruta [text]:** Identifica el nom de la ruta.
- **finished [int]:** Identifica si la ruta s'ha finalitzat (*finished* = 1) o no (*finished* = 0).
- **visibility [int]:** Identifica la privacitat de la ruta. *Visibility* = 0 indica ruta privada, *visibility* = 1 indica que només els amics la poden veure i *visibility* = 2 indica que tothom la pot veure.
- **live [int]:** Identifica si la ruta s'ha d'enviar (*live* = 1) o no (*live* = 0).
- **date [int]:** Identifica la data en què la ruta s'ha creat.
- **uploaded [int]:** Identifica si la ruta ha estat exportada al servei web (*uploaded* = 1) o no. (*uploaded* = 0).
- **total_track [int]:** Identifica el nombre de *tracks* totals. Aquest camp només s'ha utilitzat per a les proves, no té cap funcionalitat més.
- **total_dist [int]:** Identifica la distància total recorreguda en la ruta.
- **total_time [int]:** Identifica el temps total de la ruta.

La taula *track* és la que identifica un punt d'una ruta en un instant concret de temps, els seus camps són els següents:

- **id [int]:** Identificador únic del *track*.
- **sid_ruta [text]:** Clau forana que referència a la ruta a la qual pertany aquest *track*.
- **uploaded [int]:** Identifica si el *track* ha estat exportat al servei web (*uploaded* = 1) o no (*uploaded* = 0).
- **timestamp [int]:** Identifica l'instant en el qual s'ha obtingut aquest punt.

- **latitude [int]:** Identifica la latitud d'aquest punt.
- **longitude [int]:** Identifica la longitud d'aquest punt.
- **pausa [int]:** Identifica si s'ha realitzat una pausa ($pausa = 1$) o no ($pausa = 0$) en aquest punt.

6.1.2.1 DataBase

El mòdul *DataBase* només conté una classe on es defineix la base dades.

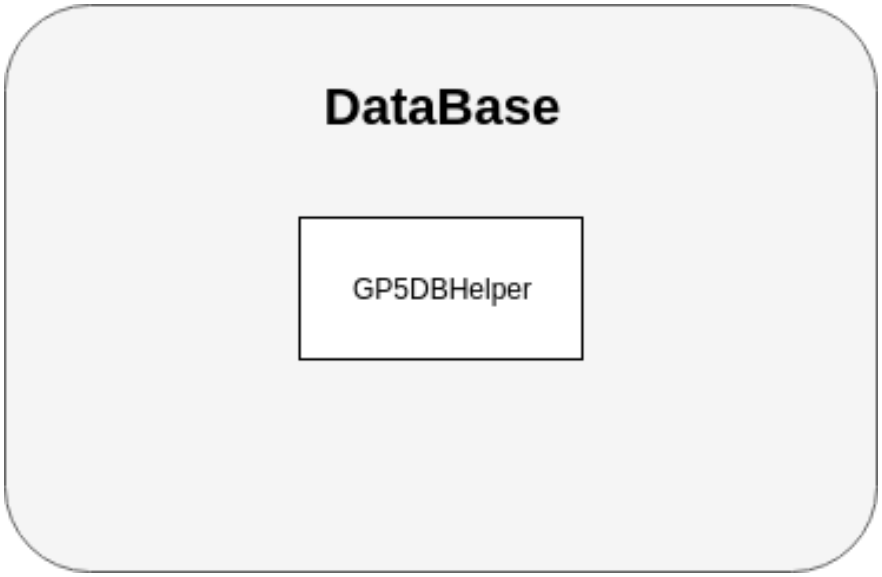


Figura 5: Mòdul DataBase.

Aquesta classe, a part de crear la base de dades, també es defineixen totes les funcions que permeten l'obtenció i manipulació de les dades. Aquestes funcions són utilitzades per la resta de mòduls.

```
public class GPSDBHelper extends SQLiteOpenHelper
{
    //inicialitzador
    public static GPSDBHelper getInstance(Context ctx);
    //definici de les taules
    public static class TaulaRutes implements BaseColumns;
```

```
public static class TaulaTrack implements BaseColumns;
//funcions
public long insertRuta(String nom_usuari,
                       String nom_ruta,
                       int visibility,
                       int live);
public void insertTrack(String sid_ruta,
                        int uploaded,
                        long timestamp,
                        double latitude,
                        double longitude,
                        int pausa);
public void insertImportedRuta(String nom_usuari,
                               String sid_ruta,
                               String nom_ruta,
                               int uploaded,
                               int finished,
                               int visibility,
                               int live,
                               long date,
                               long total_track,
                               long total_dist,
                               long total_time);


public String getSidRuta(long id);
public void deleteRuta(String sid_ruta);
```

```
    public JSONObject getFinishRuta(String sid_ruta);
    public JSONObject getRutaToExport(String sid_ruta);
    public JSONArray getTrackToExport(String sid_ruta);
    public JSONObject getNotUploadedTracks(String sid_ruta);
    public Cursor getDadesTrack(String sid_ruta);
    public int getTotalTrack(String sid_ruta);
    public Cursor getLlistaRutes(String where);
    public int getLiveRoute(String sid_ruta);
    public long getTimeRoute(String sid_ruta);
    public float getDistanceOfRoute(String sid_ruta);
    public ArrayList<GeoPoint> getDataTrack(String sid_ruta);
    public GeoPoint getLastTrack(String sid_ruta);
    public JSONObject sendPausa(String sid_ruta);
    public GeoPoint getPrevTrack(String sid_ruta);
    public ArrayList<ArrayList<GeoPoint>> getDataPrintTrack(
                                        String sid_ruta);
    public void updatePausaTrack(String sid_ruta);
    public void updateRutaFinished(String sid_ruta);
    public void updateFinished();
    public void updateRutaLiveFinished(String sid_ruta);
    public void updateTrackUploaded(long id_track);
    public void updateRutaUploaded(String sid_ruta,
                                boolean isUploaded);
    public void updatePreparedTrackUploaded(String sid_ruta,
                                        boolean isUploaded);
}
```

Les tasques de les funcions són les següents:

- **insertRuta(String nom_usuari, String nom_ruta, int visibility, int live):** Insereix una ruta a la taula ruta amb els paràmetres especificats i retorna la *id* de la ruta inserida.
- **insertTrack(String sid_ruta, int uploaded, long timestamp, double latitude, double longitude, int pausa):** Insereix un punt a la taula *tracks* amb els paràmetres especificats.
- **insertImportedRuta(String nom_usuari, String sid_ruta, String nom_ruta, int uploaded, int finished, int visibility, int live, long date, long total_track, long total_dist, long total_time):** Insereix una ruta a la taula ruta amb els paràmetres especificats. A diferència de la funció *insertRuta()* aquesta no ha de generar el *sid_ruta*.
- **getSidRuta(long id):** Retorna el camp *sid_ruta* de la ruta *id*.
- **deleteRuta(String sid_ruta):** Esborra la ruta identificada amb *sid_ruta*.
- **getFinishRuta(String sid_ruta):** Retorna un objecte *JSON* de la ruta *sid_ruta* que es vol finalitzar en un format que s'explicarà més endavant.
- **getRutaToExport(String sid_ruta):** Retorna un objecte *JSON* de la ruta *sid_ruta* que es vol exportar en un format que s'explicarà més endavant.
- **getTrackToExport(String sid_ruta):** Retorna un objecte *JSON* dels punts de la ruta *sid_ruta* que es vol exportar en un format que s'explicarà més endavant.
- **getNotUploadedTracks(String sid_ruta):** Retorna un objecte *JSON* de la llista de punts de la ruta *sid_ruta* que no s'han enviat al servei web en un format que s'explicarà més endavant.
- **getDadesTrack(String sid_ruta):** Retorna un *cursor*[6] que ens dóna accés de lectura a les dades de la taula *track* de la ruta identificada per *sid_ruta*.

- **getTotalTrack(String sid_ruta):** Retorna el nombre total de punts de la ruta identificada per *sid_ruta*.

- **getLlistaRutes(String where):** Retorna un *cursor* que ens dóna accés de lectura a les dades de les rutes que compleixen la consulta *where*.

- **getLiveRoute(String sid_ruta):** Retorna el valor corresponent al camp *live* de la ruta identificada per *sid_ruta*.

- **getTimeRoute(String sid_ruta):** Retorna en mil·lisegons el temps total de la ruta identificada per *sid_ruta*.

- **getDistanceOfRoute(String sid_ruta):** Retorna en metres la distància total de la ruta identificada per *sid_ruta*.

- **getDataTrack(String sid_ruta):** Retorna una llista de tots els punts de la ruta identificada per *sid_ruta*.

- **getLastTrack(String sid_ruta):** Retorna l'últim punt de la ruta identificada per *sid_ruta*.

- **getPrevTrack(String sid_ruta):** Retorna el penúltim punt de la ruta identificada per *sid_ruta*.

- **getDataPrintTrack(String sid_ruta):** Retorna una llista de llistes dels punts d'una ruta identificada per *sid_ruta* essent cada llista un fragment de la ruta separada per una pausa.

- **updatePausaTrack(String sid_ruta):** Actualitza el camp *pausa* a 1 de l'últim punt del *track* de la ruta identificada per *sid_ruta*.

- **updateRutaFinished(String sid_ruta):** actualitza el camp finished a 1 de la ruta identificada per sid_ruta.

- **updateFinished():** actualitza el camp finished a 1 de totes les rutes.
- **updateRutaLiveFinished(String sid_ruta):** Actualitza el camp *finished* i *uploaded* a 1 de la ruta identificada per *sid_ruta*.
- **updateTrackUploaded(long id_track):** Actualitza el camp *uploaded* a 1 del *track* identificat per *id_track*.
- **updateRutaUploaded(String sid_ruta, boolean isUploaded):** Actualitza el camp *uploaded* a 1 de la ruta identificada per *sid_ruta* si *isUploaded* és *True* altrament actualitza el camp *uploaded* a 0.
- **updateTrackUploaded(String sid_ruta, boolean isUploaded):** Actualitza el camp *uploaded* a 1 dels punts de la taula *track*, al qual *uploaded* sigui 2, de la ruta identificada per *sid_ruta* si *isUploaded* és *True* altrament actualitza el camp *uploaded* a 0.

6.1.3 Presentador

El presentador és el que realitza les comandes de l'usuari passades per la vista i es comunica amb la base de dades.

Hem separat el presentador en 4 grans blocs *GPSDataManager*, *DataTransfer*, *AsyncRequest* i *ApiWeb2py*.

6.1.3.1 GPSDataManager

El presentador *GPSDataManager* consta de tres classes que s'encarreguen de capturar, gestionar i visualitzar les dades GPS.

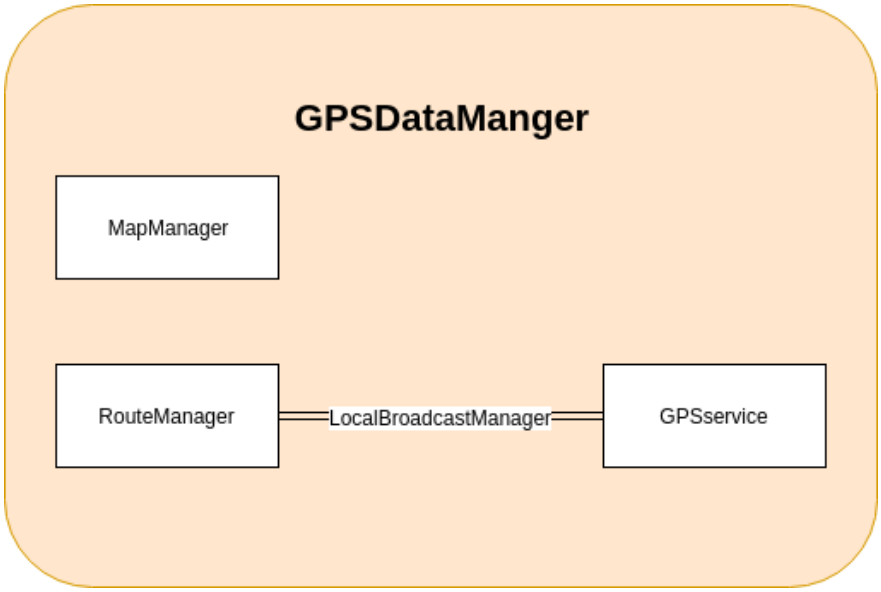


Figura 6: Mòdul GPSDataManager.

6.1.3.1.1 GPSservice

Primer de tot tenim la classe *GPSservice* que és un servei d'Android. Els serveis d'Android són un component que permeten fer tasques de llarga durada en segon pla i que no ofereixen interfície d'usuari.

La tasca que realitza aquest servei és la de inicialitzar el GPS del mòbil i crear un capturador d'events que cada cert període de temps o diferència de distància difongui el valor del GPS per un canal local utilitzant la classe, pròpia d'Android, LocalBroadcastManager.

```
public class GpsService extends Service
{
    private class LocationListener implements android.location.
        ↪ LocationListener {
        public LocationListener(String provider);
        @Override
        public void onLocationChanged(Location location);
    }
```

```
    @Override
    public void onCreate();
    @Override
    public void onDestroy();
    protected Location getBetterLocation(Location newLocation,
                                         Location currentBestLocation);
    private void initializeLocationManager();
    private void newLocationToActivity(Location location);
    private void sendLocationBroadcast(Intent intent, Location location);
}
```

La classe **LocationListener**, donat un proveïdor vincula l'objecte que ha inicialitzat la classe amb la funció *onLocationChanged()*. D'aquesta manera cada cop que el GPS rebi una nova ubicació es realitzarà una crida a aquesta funció.
Les tasques de les diferents funcions són les següents:

- **initializeLocationManager():** Aquesta funció inicialitza l'objecte LocationManager propi d'Android que és necessari per obtenir la ubicació del GPS.
- **onCreate():** Sobreescrivim aquesta funció, que es crida en inicialitzar el servei, i fem una crida a la funció *initializeLocationManager()* i inicialitzem els criteris de precisió i d'actualització del GPS.
- **onDestroy():** Aquesta funció es crida abans que es destrueixi la classe GPSservice. La sobreescrivim per tal d'esborrar les dades inicialitzades.
- **getBetterLocation(Location newLocationm, Location currentBestLocation):** Aquesta funció retorna la ubicació més precisa d'entre les dues passades.
- **sendLocationBroadcast(Intent intent, Location location):** Aquesta funció fa una difusió, a través del LocalBroadcastManager, de la ubicació.

- **newLocationToActivity(Location location):** Aquesta funció crea un intent i fa una crida a la funció *sendLocationBroadCast()*. Un intent[9] és bàsicament una estructura de dades passives que conté una descripció abstracta d'una acció a realitzar.

6.1.3.1.2 RouteManager

La classe *RouteManager* és l'encarregada d'administrar les rutes.

```
public class RouteManager
{
    //inicialitzador
    public static RouteManager getInstance(Context ctx);
    //funcions publiques
    public boolean startRoute(String routeName,
                              int visibility,
                              int live);
    public void stopRoute();
    public void pauseRoute();
    public void resumeRoute();
    public String getSIDActualRoute();
    public String getNameActualRoute();
    public int getRouteState();
    //funcions privades
    private void addTrackToRoute(long timestamp,
                                 Double latitude,
                                 Double longitude);
    private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {}
```

```
    };
    private void initializeTimerTask();
    private void startTimer();
    private void stopTimerTask();

}
```

Per inicialitzar la classe *RouteManager* s'ha de demanar una instància de la classe passant el context de l'aplicació:

```
routeManager = RouteManager.getInstance(mContext);
```

El *context* [5] en Android és informació de l'aplicació i del seu estat. És necessari donar el *context* de l'aplicació quan es creen nous objectes per tal de saber quin és l'estat actual de l'aplicació i per obtenir recursos.
Les funcions públiques que proporciona aquesta classe per administrar les rutes són les següents:

- **startRoute(String routeName, int visibility, int live):** Aquesta funció crea una nova ruta amb el nom *routeName* i la privacitat *visibility*, inicia el *GPSservice* i es registra al *LocalBroadcastManager* per rebre actualitzacions de la ubicació. En cas que *live* equivalgui a 1 inicia un temporitzador per exportar dades cada cert període de temps. Si la ruta s'ha pogut crear, retorna *True* si no retorna *False*.

- **stopRoute():** Aquesta funció cancel·la el registre en el *LocalBroadcastManager*, para el *timer* que exporta les dades, para el servei *GPSservice*, actualitza la ruta com ha finalitzada i, en cas que fos una ruta en directe, s'envia al servei web que la ruta s'ha finalitzat.

- **pauseRoute():** Aquesta funció cancel·la el registre en el *LocalBroadcastManager*, para el *timer* que exporta les dades, para el servei *GPSservice*, actualitza la ruta com a

pausada i, en cas que fos una ruta en directe, s'envia al servei web que la ruta ha estat pausada.

- **resumeRoute():** Aquesta funció inicia el servei *GPSservice*, es registra en el *LocalBroadcastManager* i si la ruta és en directe inicia el *timer* per exportar les dades.
- **getSIDActualRoute():** Aquesta funció retorna el camp *sid_ruta* de la ruta que s'està realitzant en aquell moment.
- **getNameActualRoute():** Aquesta funció retorna el nom de la ruta que s'està realitzant en aquell moment.
- **getRouteState():** Aquesta funció retorna l'estat de la ruta que s'està realitzant en aquell moment. Els estats són $START = 0$, $STOP = 1$, $PAUSE = 3$ i $RESUME = 3$.

Les funcions privades de la classe RouteManager són les següents:

- **addTrackToRoute(long timestamp, Double latitude, Double longitude):** Aquesta funció crea un nou punt de la ruta que s'està realitzant en aquell moment amb els paràmetres passats.
- **initializeTimerTask():** Aquesta funció inicialitza una funció d'exportació per a ser executada periòdicament.
- **startTimer():** Aquesta funció inicia el *timer* per a executar la funció inicialitzada a *initializeTimerTask()*.
- **stopTimerTask():** Aquesta funció para el *timer* que executa la funció inicialitzada a *initializeTimerTask()*.
- **Receptor broadcast**

```
private BroadcastReceiver mMessageReceiver = new BroadcastReceiver() {
    @Override
```

```
    public void onReceive(Context context, Intent intent) {}
};
```

Guardem a la variable local, de tipus *BroadcastReceiver*, *mMessageReceiver* al receptor del *LocalBroadcastManager* i sobreescrivim la funció *onReceive* per gestionar les dades que rebem. En la funció *onReceive* rebem la nova ubicació passada pel *GPSservice* i afegim un nou punt cridant a la funció *addTrackToRoute*.

6.1.3.1.3 MapManager

La classe MapManager és l'encarregada de gestionar la visualització de les rutes en el mapa.

```
public class MapManager
{
    //inicialitzador
    public static MapManager getInstance(Context ctx);
    //funcions
    public void setMapView(MapView view);
    public void setDefaultConfig();
    public void setOwnConfig(Integer road_color,
                             Integer road_width,
                             ITileSource tileSource,
                             Boolean multiTouchControls,
                             Boolean rotationGestureOverlay,
                             Integer defaultZoom,
                             GeoPoint defaultCenter);
    public void setToNewPosition(GeoPoint newPosition);
    public void setMarkerToPosition(GeoPoint position,
                                    int marker_type);
    public void printFinalRoute(String sid_ruta);
```

```
    public void printAllRoute(String sid_ruta);
    public void updateRoute(String sid_ruta);
    public BoundingBox computeArea(String sid_ruta);
    public void clearMap();
    public void setMyLocationMarker(GeoPoint my_position);
    public void removeMyLocationMarker();
}
```

Les funcions públiques que proporciona aquesta classe per administrar la visualització de les rutes són les següents:

- **setMapView(MapView view):** Aquesta funció enllaça la vista *view* amb la classe *MapManager*, inicialitza el controlador del mapa i la icona/marcador de la ubicació.
- **setDefaultConfig():** Aquesta funció estableix els valors per defecte del color i l'amplada de la ruta, el zoom, el punt central del mapa en inicialitzar-lo, la font dels mapes, la funcionalitat de fer zoom pressionant dos cops i la possibilitat de rotar el mapa.
- **setOwnConfig(Integer road_color, Integer road_width, ITileSource tileSource, Boolean multiTouchControls, Boolean rotationGestureOverlay, Integer defaultZoom, GeoPoint defaultCenter):** Aquesta funció permet configurar els paràmetres mencionats en la funció *setDefaultConfig()*.
- **setToNewPosition(GeoPoint newPosition):** Aquesta funció centra el mapa a la posició indicada pel *GeoPoint newPosition*. Un *GeoPoint* és un punt identificat a partir d'una latitud i d'una longitud.
- **setMarkerToPosition(GeoPoint position, int marker_type):** Aquesta funció dibuixa una icona identificada per *marker_type* a la posició *position* en el mapa. Hi ha quatre icones diferents que identifiquen l'inici, una parada, una represa de la ruta i el final.

- **printFinalRoute(String sid_ruta)** i **printAllRoute(String sid_ruta):** Aquestes dues funcions fan exactament el mateix, dibuixen tota la ruta al mapa, però amb la diferència que el *printFinalRoute()* afegeix les icones d'inici, parada, represa i final mentre que el *printAllRoute()* no.

- **updateRoute(String sid_ruta):** Aquesta funció actualitza la ruta *sid_ruta* amb els nous punts i centra el mapa a l'últim punt.

- **computeArea(String sid_ruta):** Aquesta funció calcula un rectangle que correspon a l'àrea de la ruta identificada per *sid_ruta*.

- **clearMap()**: Aquesta funció esborra tots els dibuixos i icones del mapa.

- **setMyLocationMarker(GeoPoint my_position)** i **removeMyLocationMarker():** Aquestes dues funcions gestionen la icona de la ubicació actual. No es fa servir la funció *setMarkerToPosition()* perquè necessitem tenir identificat en tot moment la icona d'ubicació per tal d'anar-la esborrant i afegint. La funció *setMyLocationMarker* situa la icona d'ubicació a la posició *my_position* i la funció *removeMyLocationMarker* l'esborra.

6.1.3.2 AsyncRequests

Aquestes classes s'encarreguen de gestionar la comunicació entre l'aplicació mòbil i el servei web.

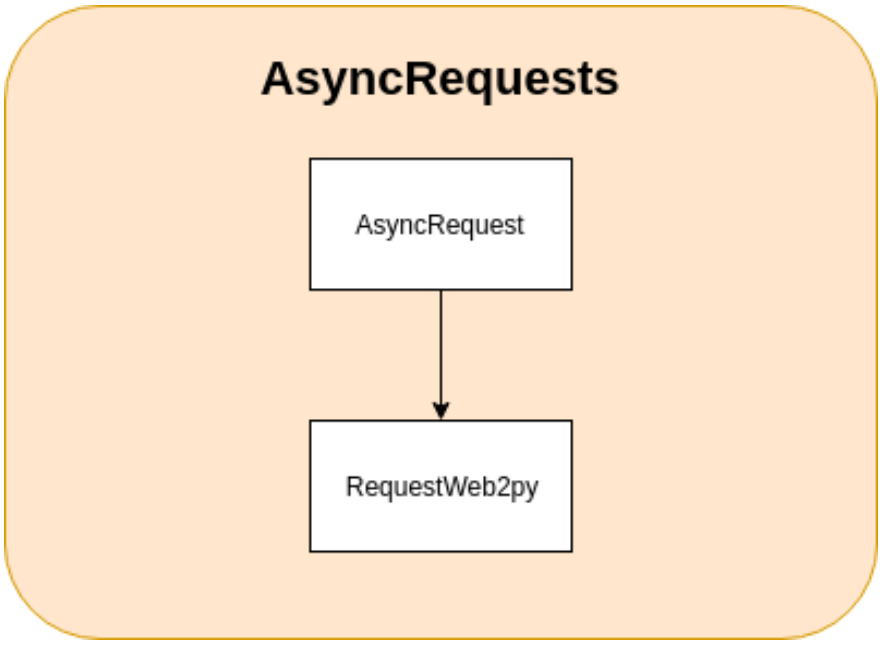


Figura 7: Mòdul AsyncRequests.

6.1.3.2.1 AsyncRequest

La classe *AsyncRequest* estén la classe *AsyncTask*[4]. Aquesta classe permet realitzar tasques en segon pla i actualitzar la interfície d'usuari un cop la tasca ha finalitzat.

```
public class AsyncRequest extends AsyncTask<String, String, String>
{
    //incialitzador
    public AsyncRequest(Context context,
                        int method,
                        String post_url,
                        JSONObject data,
                        InterfacesW2P interfacesW2P);
    @Override
```

```
    protected String doInBackground(String... params);
    @Override
    protected void onPostExecute(String result);
}
```

A l'inicialitzador de la classe es passa els paràmetres necessaris per a realitzar una petició al servei web. Com podem veure hi ha el paràmetre *InterfacesW2P* aquest paràmetre és una interfície que consta de diferents funcions que serveixen per realitzar un *callback* allà on han estat definides.

L'extensió de la classe *AsyncTask* proporciona diferents funcions, nosaltres en sobreescrivim les dues que necessitem:

- **doInBackground(String... params):** Aquesta funció és la que realitza les tasques en segon pla. Aquí, segons els paràmetres passats en la inicialització de la classe *AsyncRequests*, fem una crida a una funció de la classe *ApiWeb2py*.
- **onPostExecute(String... params):** Aquesta funció s'executa un cop s'obté un resultat de la crida a la funció de la classe *ApiWeb2py*. En aquesta funció executem un *callback* per avisar a la línia d'execució principal que s'ha obtingut una resposta.

6.1.3.2.2 RequestWeb2py

La classe *RequestWeb2py* és la que s'encarrega de gestionar la comunicació entre l'aplicació mòbil i el servei web. Per fer les peticions http hem utilitzat la llibreria *volley*[16].

```
public class RequestWeb2Py {
{
    public String post(Context context,
                       String post_url,
                       final JSONObject params);
    public String get(Context context,
```

```
                    String get_url);
    private boolean getNewToken();
    private String theRequest(int method,
                              String post_url,
                              final JSONObject params);
    public String login(Context context,
                        String login_url,
                        String username,
                        String pwd);
    private boolean tryLogin(String username, String pwd, String url);
}
```

Les tasques que realitzen les funcions d'aquesta classe són les següents:

- **post(Context context, String post_url, final JSONObject params)** i **get(Context context, String get_url):** Aquestes dues funcions abans d'iniciar la comunicació amb el servei web comproven si tenen algun *token* i en cas que en tinguin comproven si està caducat. En cas que no es tingui cap *token* o estigui caducat se sol·licita un nou *token* cridant a la funció *getNewToken()*. Un cop obtenim un *token* cridem a la funció *theRequest()* per dur a terme la petició. La diferència entre la funció *post* i la funció *get* és que la funció *post* només retorna el codi de resposta HTTP mentre que la funció *get*, si no s'ha produït cap error, retorna un *string* amb les dades obtingudes.

- **getNewToken():** Aquesta funció fa una petició a un URL especifica del servei web amb el nom d'usuari i la contrasenya de l'usuari, per tal d'obtenir un *token* per evitar la sobrecàrrega del servei web. En cas que s'obtingui un *token* amb èxit retorna *True* si no retorna *False*.

- **theRequest(int method, String post_url, final JSONObject params):** Aquesta

funció fa una petició de tipus *method*, que pot ser POST o GET, a l'URL *post_url* amb els paràmetres *params* i retorna la resposta obtinguda.

- **tryLogin(String username, String pwd, String url):** Aquesta funció intenta iniciar sessió al servei web amb l'usuari *username* i la contrasenya *pwd*. Si s'inicia sessió amb èxit retorna *True* si no retorna *False*.

- **login(Context context, String login_url, String username, String pwd):** Aquesta funció crida a *tryLogin()* i en cas de que retorni *True* retorna el codi d'èxit HTTP i en cas que retorni *False* retorna el codi d'error HTTP corresponent.

6.1.3.3 DataTransfer

Aquest mòdul inclou les classes que s'encarreguen de la importació i exportació de dades amb el servei web. Aquestes classes utilitzen el mòdul *AsyncRequests* per a comunicar-se amb el servei web.

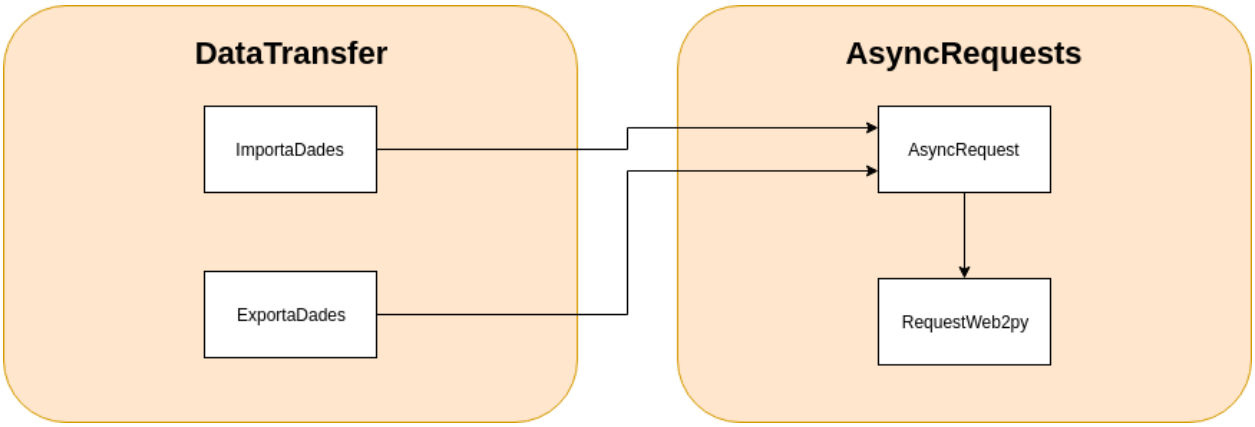


Figura 8: Mòdul DataTransfer.

Abans d'explicar les diferents classes i les seves funcions d'aquest mòdul hem de definir les normes de comunicació entre el servei web i l'aplicació mòbil.

6.1.3.3.1 Comunicació aplicació mòbil-servei web

Les dades entre aplicació i el servei web es passen en format *JSON*[10], ja que és un format fàcil de generar i analitzar.

Les dades que es volen enviar i rebre són les dades de les rutes i els seus punts, d'aquesta manera s'ha establert els següents formats d'enviament d'informació des de l'aplicació al servei web:

- **Ruta:** El format d'enviament d'una ruta és el següent:

```
{'ruta': {
        'total_time': '',
        'nom_ruta': '',
        'id_ruta': '',
        'visibility': '',
        'live': '',
        'total_track': '',
        'data': '',
        'total_dist': '',
        'track': [{
          'latitude': '',
          'longitude': '',
          'timestamp': '',
          'id_ruta': '',
          'pausa': ''
        }]
    }
}
```

- **Track:** El format d'enviament d'un *track* és en forma de llista:

```
{'track': [{
      'latitude': '',
      'longitude': '',
```

```
        'timestamp': '',
        'id_ruta': '',
        'pausa': ''
      }]
  }
```

- **Iniciar:** Per iniciar una ruta en directe es fa servir el mateix format que per enviar una ruta, però sense el *track*, el *total_track*, el *total_dist* i el *total_time*.

- **Finalitzar:** Per marcar una ruta en directe com a finalitzada es fa servir el mateix format que per enviar una ruta, però sense el *track*.

- **Pausa:** Per marcar una pausa en un punt es passa la informació del punt específic de la següent manera:

```
  { 'id_ruta': '',
    'latitude': '',
    'longitude': '',
    'timestamp': '',
    'pausa': '1'
  }
```

Des de l'aplicació es vol rebre les rutes que hi ha al servei web. El servei web envia les dades i respon a les peticions HTTP de la següent manera:

- **Reposta:** Les respostes que envia el servei web a les peticions HTTP segueixen sempre el següent format:

```
  {'response': 'codi http'}
```

- **Enviament de rutes:** Quan el servei web rep una petició *GET* demanen que retorni les rutes, retorna una llista de *JSONs* per cada ruta on cada ruta té una llista del seu *track*. El format és el següent:

```
{"response":
  [
    { "total_time": '',
      "nom_ruta": '',
      "id_ruta": '',
      "finished": '',
      "visibility": '',
      "total_track": '',
      "live": '',
      "id_usuari": '',
      "data": '',
      "id": '',
      "total_dist": '',
      "track":[
        { "latitude": '',
          "timestamp": '',
          "longitude": '',
          "pausa": ''
        }
      ]
]}
```

6.1.3.3.2 ExportaDades

La classe *ExportaDades* consta de diverses funcions que permeten enviar informació de les rutes al servei web.

```
public class ExportaDades
{
    //inicialitzador
    public ExportaDades(Context context);
    //funcions publiques
    public void exportaRuta(final String sid_ruta,
                            final InterfaceExporter interfaceExporter);
    public void estateMachine(String sid_ruta);

    public void exportLiveRoute(final String sid_ruta);
    public void finishRuta(final long id_ruta,
                           final InterfaceExporter interfaceExporter);
    public void pauseRuta(final String sid_ruta);
}
```

Les tasques de les funcions són les següents:

- **exportaRuta(final String sid_ruta, final InterfaceExporter interfaceExporter):** Aquesta funció obté tota la informació de la ruta identificada per *sid_ruta* i envia les dades al servei web. Quan obté una resposta fa un *callback* utilitzant les funcions definides per la interfície *interfaceExporter*.

- **exportLiveRoute(final String sid_ruta):** Aquesta funció inicia una ruta en directe enviant la informació de la ruta al servei web i enviant els punts que encara no han estat enviats.

- **estateMachine(String sid_ruta):** L'enviament de les dades de les rutes en directe es realitza a través d'una màquina d'estats:

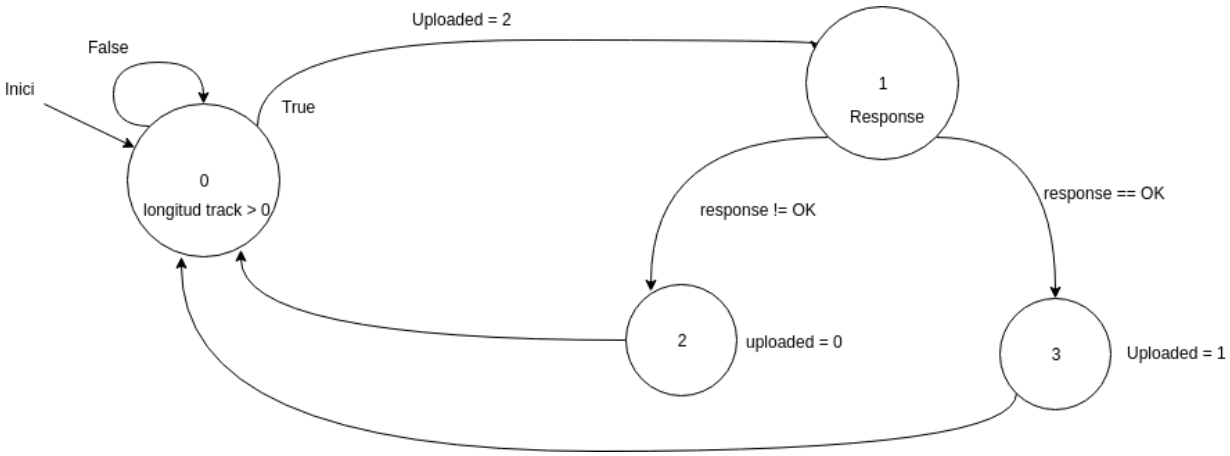


Figura 9: Màquina d'estats per l'enviament de dades en directe.

En **l'estat 0** s'afegeixen tots els punts de la ruta que s'està fent el directe, els quals el seu camp *uploaded* valgui 0, en una llista i s'actualitza el seu valor a 2. Si el nombre de punts és superior a 0, passem a l'estat 1.

En **l'estat 1** enviem els *punts* al servidor i esperem una resposta. Si la resposta és "OK"vol dir que els punts s'han enviat amb èxit i canviem a l'estat 3. Si rebem una resposta diferent de "OKçanviem a l'estat 2.

En **l'estat 2** tots els punts amb *uploaded* a 1 els actualitzem a *uploaded* a 0, ja que no s'han enviat correctament i tornem a l'estat inicial 0.

En **l'estat 3** tots els punts amb *uploaded* a 2 els actualitzem a *uploaded* a 1, ja que s'han enviat correctament i tornem a l'estat inicial 0.

- **finishRuta(final long id_ruta, final InterfaceExporter interfaceExporter):** Aquesta funció avisa al servidor què una ruta ha finalitzat.

- **pauseRuta(final String sid_ruta):** Aquesta funció avisa al servidor que la ruta en directe ha estat pausada.

6.1.3.3.3 ImportaDades

La classe *ImportaDades* consta de diverses funcions que permeten la recepció de les rutes que hi ha en el servei web.

```
public class ImportaDades
{
    //inicialitzador
    public ImportaDades(Context context);
    //funcions
    public void importarRutes(final InterfaceExporter interfaceExporter);
    private void manageImportedRoutes(String rutes);
}
```

Les tasques de les funcions són les següents:

- **importarRutes(final InterfaceExporter interfaceExporter):** Aquesta funció utilitza el mòdul *AsyncRequests* per fer una petició *GET* al servidor per obtenir les rutes. Quan rep la resposta, si no s'ha produït cap error, es crida la funció *manageImportedRoutes()* i es crida a una de les funcions de l'*interfaceExporter* per avisar que ja s'ha acabat la importació. En cas que es produeixi un error es fa una crida a una de les funcions de l'*interfaceExporter()* per avisar que s'ha produït un error.

- **manageImportedRoutes(String rutes):** Aquesta funció s'encarrega d'analitzar i processar la resposta de la petició *GET* i insereix les rutes que no existissin en l'aplicació mòbil.

6.1.4 Vista

La vista s'encarrega de proporcionar una interfície gràfica per representar les dades i permetre a l'usuari interactuar amb l'aplicació.

La vista està composta per una *activity*[1] i el seu corresponent *layout*[11]. L'*activity* és la capa representativa de l'aplicació que veu l'usuari. El *layout* defineix l'estructura de la interfície de l'*activity*.

La nostra aplicació està formada bàsicament per sis *activities* amb els seus corresponents *layouts*.

6.1.4.1 Login

L'*activity Login* és la pantalla inicial que ens apareix en executar l'aplicació en cas que encara no hàgim iniciat la sessió, veure [10].

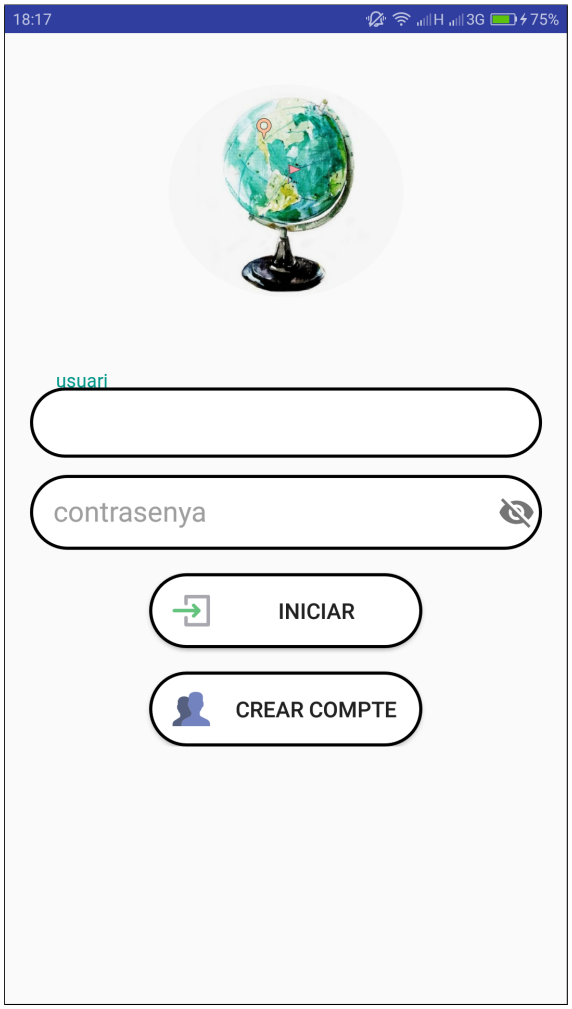


Figura 10: Activity login

En aquesta activitat podem dur a terme dues accions diferents que ens portaran a una o altra *activity*. La primera acció és la d'iniciar sessió que en cas d'èxit ens portarà a l'*activity Menu*. La segona acció és la de crear un compte el qual ens portarà a l'*activity WebView*.

Un cop iniciada la sessió es guardaran les dades d'usuari i contrasenya a les *SharedPreferences*[15]. D'aquesta manera quan tornem a executar l'aplicació no entrarem a l'*activity Login* sinó que anirem directament a l'*activity Menu*. Les *SharedPreferences* permeten guardar dades identificades per un parell clau valor que persistiran fins i tot quan finalitzem l'execució de l'aplicació.

6.1.4.2 WebView

L'*activity WebView* és una pantalla que fa de navegador web i ens portà a la pàgina de crear compte del servei web, veure [11].

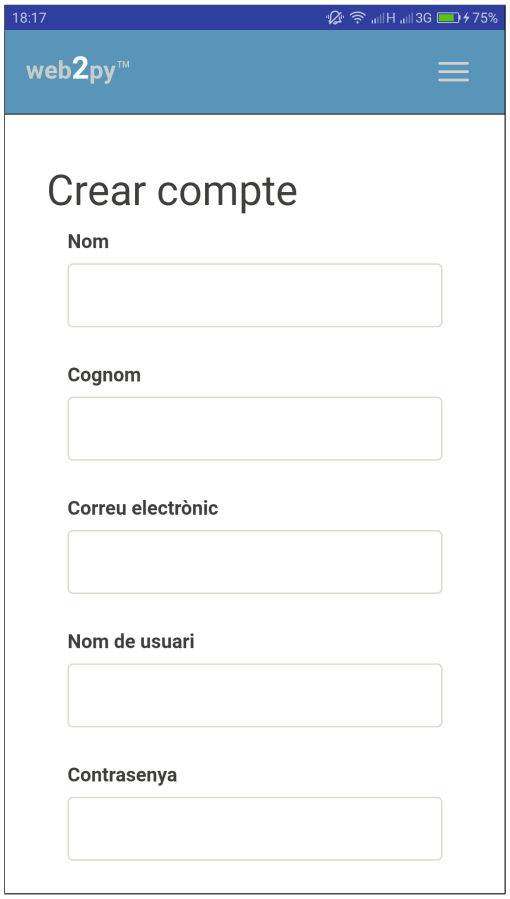


Figura 11: Activity WebView

En aquesta activitat creem el compte del servei web. Un cop creat el compte tornarem a l'activitat *Login*. Un cop confirmat l'e-mail de verificació de creació de compte ja podríem iniciar sessió a l'aplicació.

6.1.4.3 Menu

L'*activity Menu* és la pantalla principal un cop hem iniciat sessió, veure [12].

Figura 12: Activity Menu

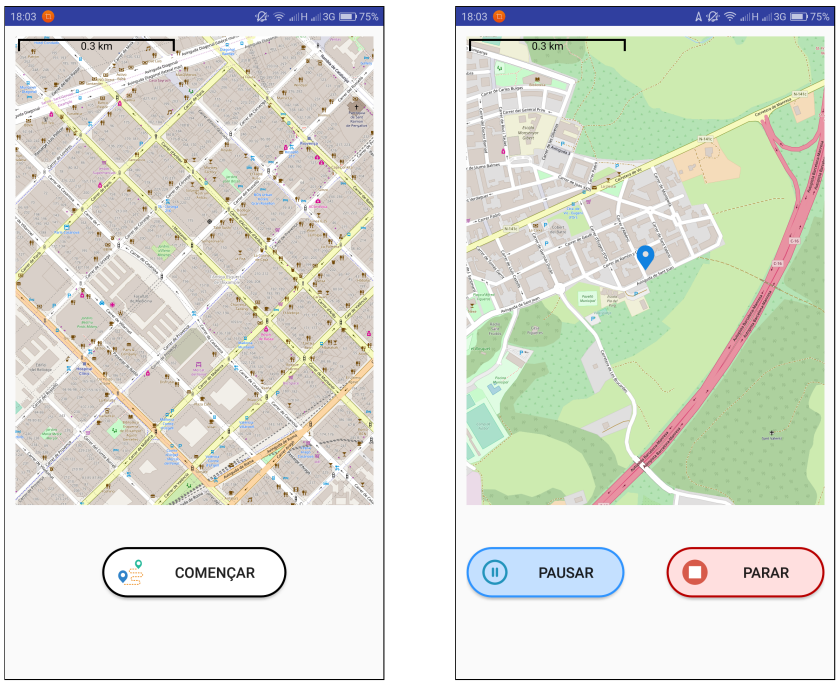
En aquesta activitat podem realitzar cinc accions diferents:

- **Iniciar Ruta:** Si premem el botó iniciar ruta, ens dirigirem a l'*activity IniciarRuta*.

- **Veure Rutes:** Si premem el botó veure rutes, ens dirigirà a l'*activity LlistaRutes*.
- **Importar:** Si premem el botó importar, i no hi ha cap ruta iniciada, ens demanarà si volem importar les rutes del servei web. En cas afirmatiu les importarà.
- **Sortir:** Si premem el botó sortir, esborrarà les dades guardades a les *SharedPreferences* referents a l'usuari i ens dirigirà a l'*activity Login*.
- **Configuració:** El botó de configuració actualment no fa res i està pensat per línies futures.

6.1.4.4 IniciarRuta

En l'*activity IniciarRuta* és on es creen les rutes, veure [13a]. En prémer el botó començar se'ns demanarà el nom de la ruta, la privacitat i si volem enviar la nostra ubicació. Un cop completat tots els camps s'iniciarà una ruta, veure [13b].



(a) Ruta abans de començar (b) Ruta començada

Figura 13: Activity IniciarRuta

Un cop iniciada una ruta es pot aturar premen el botó aturar o es pot finalitzar premen el botó parar.

6.1.4.5 LlistaRutes

En l'*activity LlistaRutes* podem veure totes les rutes que hem realitzat amb el seu nom, el temps total, la distància total i la velocitat mitjana, veure [13b].

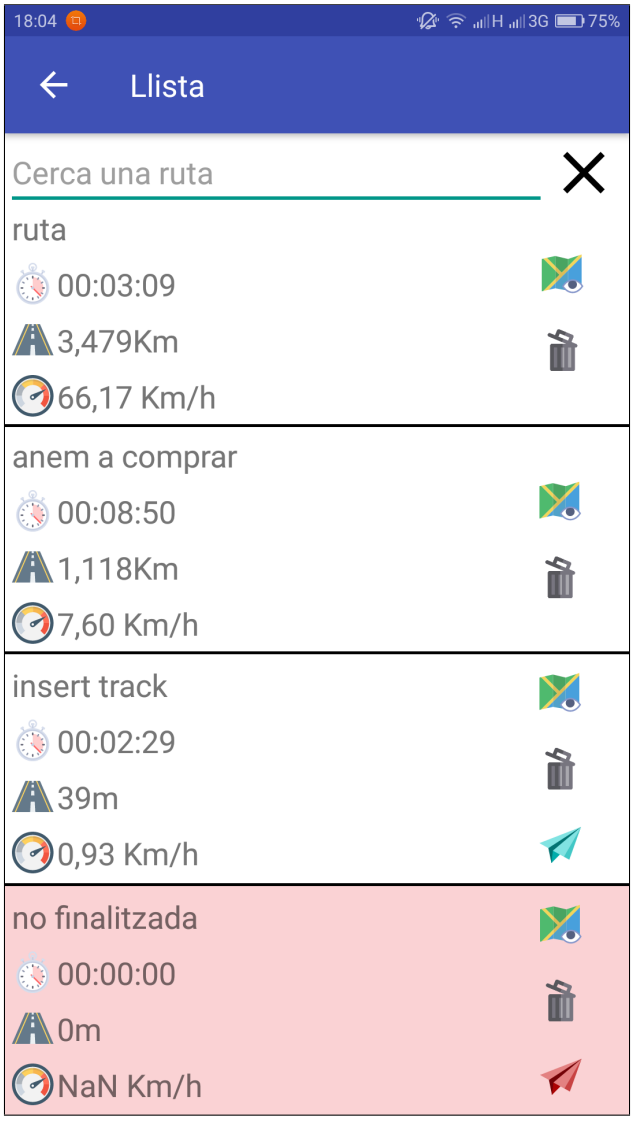


Figura 14: Activity LlistaRutes

Com podem veure totes les rutes tenen un mínim de dos botons, el botó del mapa permet veure la ruta anant a l'*activity ShowMap* de la ruta en concret i el botó d'esborrar ruta que ens

permet esborrar la ruta en concret. Finalment hi ha un altre botó amb dos colors diferents. El blau identifica que la ruta no s'ha exportat i si el premem, ens demanarà confirmació per exportar-la. El vermell identifica que la ruta s'estava fent en directe i que, per alguna raó, no s'ha finalitzat, si premem el botó, ens demanarà confirmació per finalitzar-la.

6.1.4.6 ShowMap

En l'*activity ShowMap* mostrem les rutes finalitzades amb les icones per marcar les aturades i l'inici i fi de la ruta, veure [15].

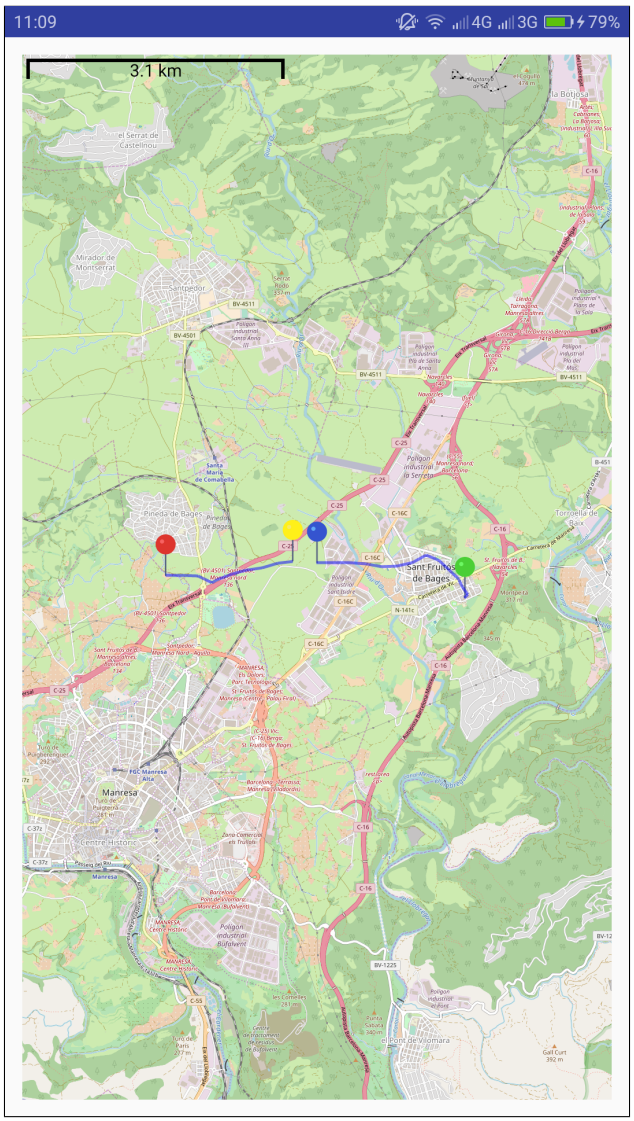


Figura 15: Activity ShowMap

En color verd tenim la icona que indica l'inici de la ruta, en color blau la icona que indica una aturada, en color groc la icona que indica una represa de la ruta i finalment en color vermell la icona que indica el final de la ruta.

6.2 Aplicació web

6.2.1 Estructura aplicació web

Web2py segueix l'arquitectura model-vista-controlador (MVC), veure [16].

La **vista** és la interfície de l'aplicació web, és la responsable de rebre les accions de l'usuari i de presentar les dades.

El **model** representa les dades que l'aplicació web processa en temps d'execució i la lògica per processar aquestes dades.

El **controlador** obté l'acció de l'usuari de la vista i ho converteix en comandes pel model o la vista.

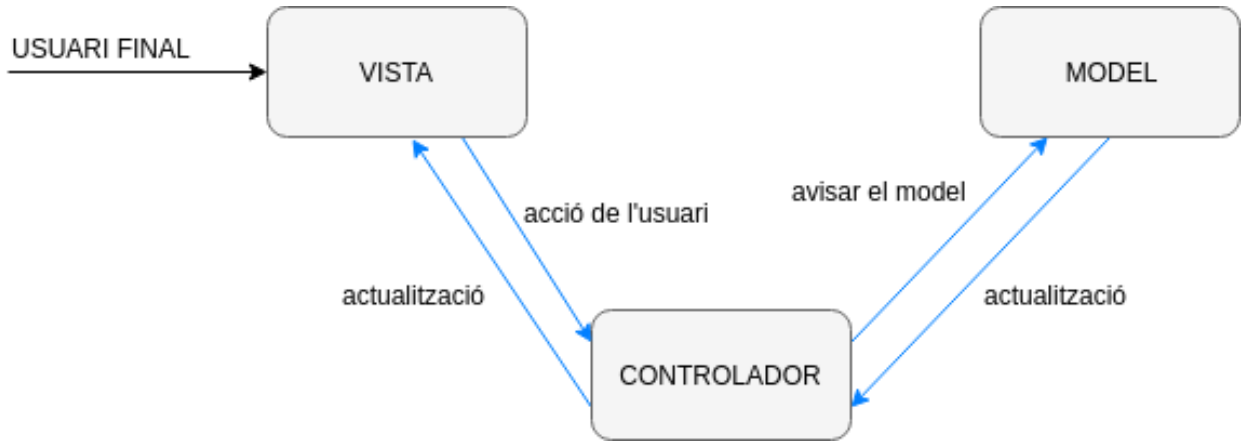


Figura 16: Arquitectura model-vista-controlador

A partir de l'arquitectura hem desenvolupat l'estructura del servei web, veure [17].

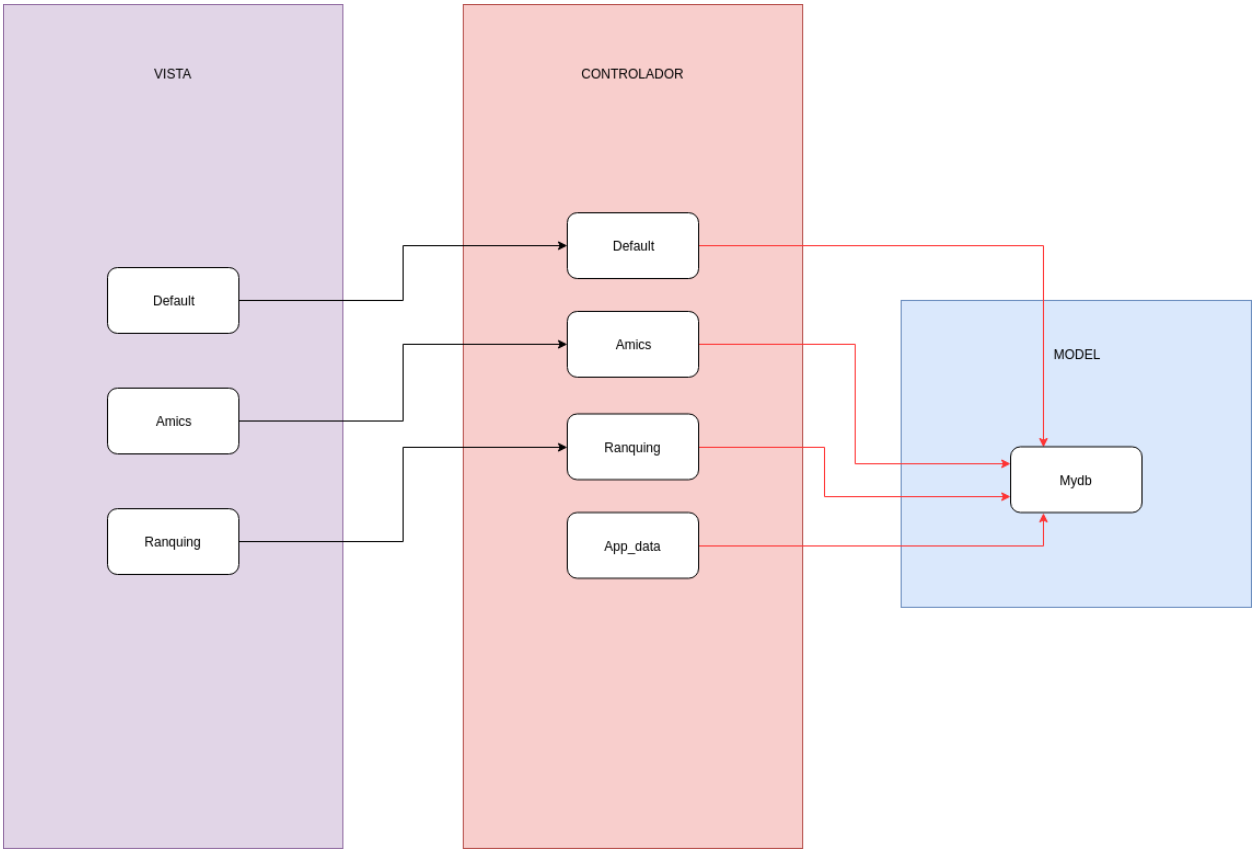


Figura 17: Estructura servei web

6.2.1.1 Model

En el model definim la nostre base de dades, veure [18]

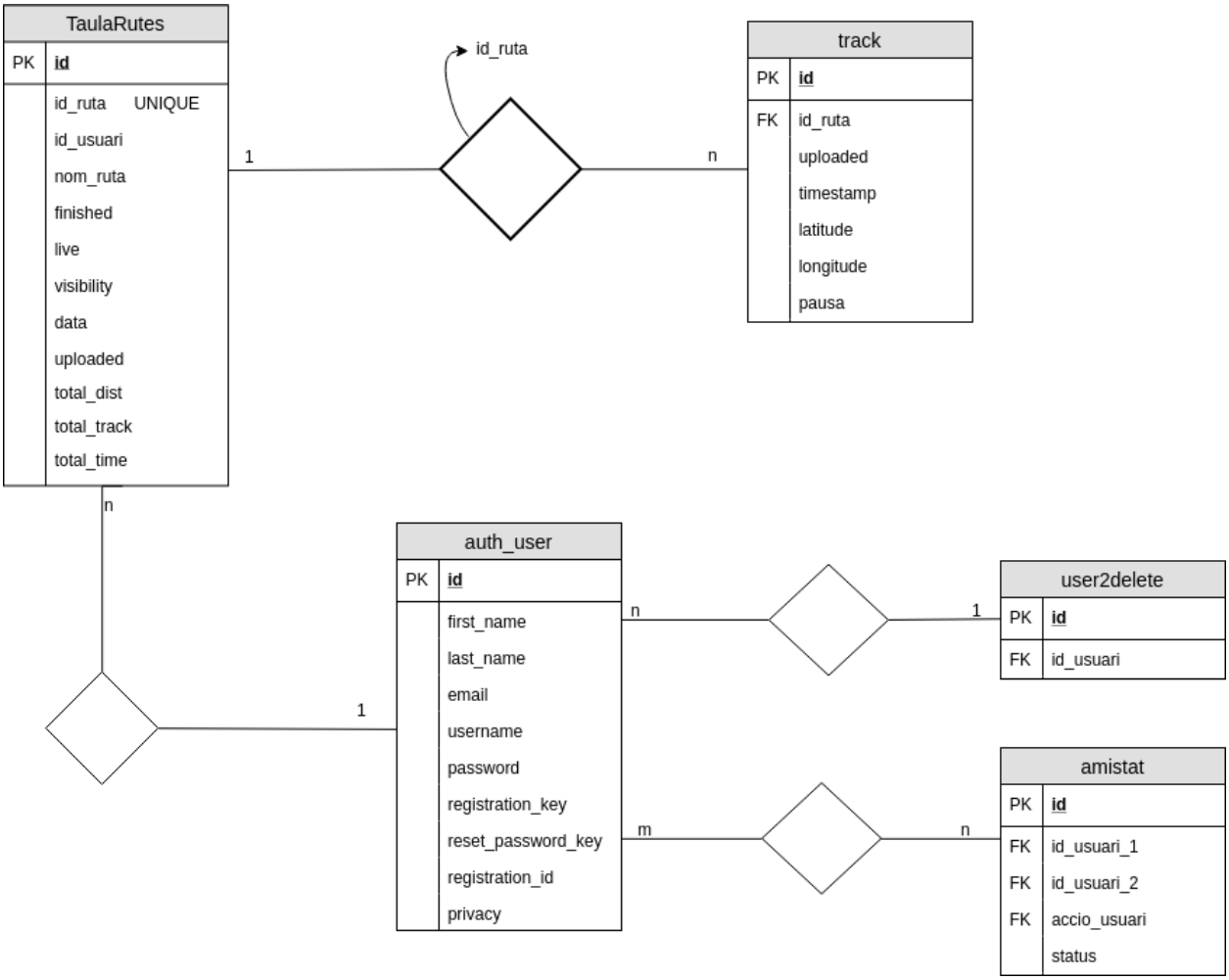


Figura 18: Base de dades servei web

Aquestes són les principals taules de la base de dades del model de la nostre pàgina web.

auth_user

La taula *auth_user* és l'encarregada d'identificar els usuaris del servei web. Aquesta taula és creada automàticament per *web2py* amb tots els camps propis per identificar un usuari. Tot i això s'ha afegit el camp *privacy* per tal que l'usuari pugui controlar qui vol que vegi les seves dades.

TaulaRutes

La *TaulaRutes* és la taula que identifica les rutes. Té els mateixos camp que la *TaulaRutes* de l'aplicació mòbil on el camp *sid_usuari* és el camp *id_usuari*.

TaulaTrack

La *TaulaTrack* és la taula que identifica els punts d'una ruta. Té els mateixos camps que la *TaulaTrack* de l'aplicació mòbil on el camp *sid_usuari* és el camp *id_usuari*.

amistat

La taula *amistat* identifica la relació d'amistat entre dos usuaris i quin usuari ha realitzat l'acció. El camp *accio_usuari* identifica l'usuari que ha realitzat l'acció i el camp *status* identifica l'estat de la relació, pendent, acceptat, rebutjat o bloquejat.

user2delete

La taula *user2delete* s'encarrega de registrar els usuaris que volen esborrar les dades, per tal d'esborrar-los un cop tanquin la sessió.

6.2.1.2 Vista

La vista esta formada per tres mòduls diferents.

6.2.1.2.1 Default

Aquest mòdul conté tots les vistes relacionades amb les rutes, la pàgina d'inici i la pàgina de configuració de l'usuari.

Aquestes són les vistes del mòdul default:

- **Index:** Aquesta vista mostra dues opcions, veure rutes o veure amics.
- **Main:** Aquesta vista permet elegir entre veure les teves rutes, veure les rutes en directe o veure el rànquing.

- **User:** En aquesta vista podem configurar les nostres dades d'usuari i fins i tot esborrar l'usuari.
- **routeList:** aquesta vista ens proporciona una llista de rutes. Depenent dels paràmetres passats ens mostrarà una llista de les nostres rutes o de les rutes d'un amic en concret.
- **LiveRouteList:** Aquesta vista ens mostra una llista de les rutes que s'estan realitzant en directe.
- **veure_ruta:** Aquesta vista ens permet veure una ruta específica.
- **display_form_ruta:** Aquesta vista ens permet modificar la privacitat i el nom d'una ruta i fins i tot esborrar-la.
- **veure_ruta:** Aquesta vista ens permet veure el mapa de la ruta especificada.
- **veure_live_ruta:** Aquesta vista ens permet veure el mapa en directe de la ruta especificada i s'actualitza cada cinc segons.

6.2.1.2.2 Amics

Aquest mòdul conté totes les vistes relacionades amb la gestió de les amistats del servei web. Aquestes són les vistes del mòdul amics:

- **Main:** En aquesta vista podem elegir entre veure la llista dels nostres amics, la llista dels usuaris bloquejats, buscar amics o veure les nostres peticions d'amistat.
- **amicsList:** En aquesta vista podem veure una llista de tots els nostres amics. Podem eliminar, bloquejar i fins i tot veure les rutes dels amics.
- **blockedList:** En aquesta vista podem veure els usuaris que hem bloquejat i, si volem, els podem desbloquejar.
- **buscarAmics:** En aquesta vista podem buscar amics que no tinguem agregats o bloquejats i els podem bloquejar o enviar una petició d'amistat.

- **peticionsAmistat:** Aquesta vista mostra les peticions d'amistat que hem rebut. Podem acceptar, rebutjar o bloquejar.

6.2.1.2.3 Rànquing

Aquest mòdul només conté una vista:

- **Main:** En aquesta vista podem veure el rànquing de kilòmetres recorreguts i de velocitat mitjana entre els nostres amics i entre tots els usuaris.

6.2.1.3 Controlador

Tenim un total de quatre mòduls controladors.

6.2.1.3.1 Default

Aquest mòdul controlador està lligat amb el mòdul default de les vistes. Les funcions d'aquest controlador són les següents:

- **index():** Només uneix l'URL amb la corresponent vista.
- **main():** Només uneix l'URL amb la corresponent vista.
- **routeList():** Proporciona a la vista *routeList* una llista de rutes i *ids* de ruta.
- **liveRouteList():** Proporciona a la vista *liveRouteList* una llista de rutes i *ids* de ruta que s'estant realitzant en directe.
- **veure_ruta():** Proporciona a la vista *veure_ruta* el *track* de la ruta en concret.
- **veure_live_ruta():** Proporciona a la vista *veure_live_ruta* el *track* de la ruta en concret i l'URL per obtenir el *track* actualitzat.
- **display_form_ruta():** Proporciona a la vista *display_form_ruta* el formulari a mostrar.
- **user():** Proporciona a la vista *user* el formulari per modificar o esborrar l'usuari.

- **update_live_ruta():** Retorna el *track* de la ruta especificada en format *JSON*.
- **update_live_list():** Retorna una llista de rutes en directe en format *JSON*.
- **update_ruta_list():** Retorna una llista de rutes, depenent dels paràmetres passats, en format *JSON*.
- **get_all_track(id_ruta):** Retorna el *track* de la ruta especificada per *id_ruta*.
- **get_list_rutes():** Retorna una llista de les rutes de l'usuari que crida la funció.
- **get_list_rutes_friend(friend):** Retorna una llista de les rutes de l'usuari *friend* amb visibilitat per a amics o per a tothom.
- **get_name_friend(friend):** Donat una *id friend* retorna el nom de l'usuari identificat per aquesta *id*, sempre que hi hagui una relació d'amistat.
- **get_list_lives_rutes():** Retorna la llista de rutes que s'estan realitzant en directe les quals l'usuari que ha cridat la funció en té accés.
- **delete_user():** Afegeix a la taula *user2delete* l'*id* de l'usuari que ha cridat la funció.
- **step2logout():** Edita el redreçament de quan sortim del compte perquè cridi la funció *my_log_out()*, un cop editat tanca la sessió de l'usuari.
- **my_log_out():** Esborra les dades de tots els usuaris de la taula *user2delete*.
- **is_friend(friend):** Retorna *True* si l'usuari que ha cridat la funció i l'usuari identificat per l'identificador *friend* són amics, altrament retorna *False*.
- **check_permission_route(user,route):** Retorna *True* si l'usuari que ha cridat la funció té permisos per veure la ruta, ja sigui perquè és propietari de la ruta o perquè la ruta és propietat d'un amic que ha donat permisos perquè els seus amics la puguin veure, altrament retorna *False*.

- **metert2km(meters):** Retorna un text convertint els metres, passats a la variable *meters*, en quilometres sempre que sigui més gran de 1000.

- **time2hours(millis):** Retorna un text en format hores:minuts:segons de la conversió dels mil·lisegons passats a la variable *millis*.

6.2.1.3.2 Amics

Aquest mòdul controlador està lligat amb el mòdul *Amics* de les vistes.

Les funcions d'aquest controlador són les següents:

- **main():** Només uneix l'URL amb la corresponent vista.

- **amicsList():** Proporciona a la vista *amicsList* una llista dels amics de l'usuari que l'ha cridada.

- **peticionsAmistat():** Proporciona a la vista *peticionsAmistat* una llista dels usuaris que han realitzat una petició d'amistat a l'usuari que ha cridat la funció.

- **blocked_list():** Proporciona a la vista *blocked_list* una llista dels usuaris que ha bloquejat l'usuari que ha cridat la funció.

- **buscarAmics():** Proporciona a la vista *buscarAmics* els URL corresponents per afegir o bloquejar usuaris.

- **getFriends(usuari):** Retorna els amics en forma de diccionari de l'usuari *usuari*.

- **checkStatus(usuari1, usuari2, status):** Retorna *True* si l'estatus entre *usuari1* i *usuari2* és igual a *status* i l'*usuari1* és el responsable de l'acció.

- **response_petition():** Respon a una petició d'amistat depenent dels paràmetres passats a través de l'URL. Si tot ha anat correctament, respon amb el codi HTTP 200, altrament respon amb el codi HTTP 400.

- **delete_or_block_friend():** Esborra o bloqueja un amic concret obtingut dels paràmetres passats a través de l'URL. Si tot ha anat correctament, respon amb el codi HTTP 200, altrament respon amb el codi HTTP 400.

- **add_or_block_user():** Esborra o bloqueja a un usuari concret obtingut dels paràmetres passats a través de l'URL. Si tot ha anat correctament, respon amb el codi HTTP 200, altrament respon amb el codi HTTP 400.

- **unblock_user():** Desbloqueja a un usuari concret obtingut dels paràmetres passats a través de l'URL. Si tot ha anat correctament, respon amb el codi HTTP 200, altrament respon amb el codi HTTP 400.

- **friendship_exist(usuari1, usuari2):** Retorna *True* si hi ha alguna relació entre *usuari1* i *usuari2*, altrament retorna *False*.

- **delete_friendship(usuari1, usuari2):** Esborra qualsevol relació entre l'*usuari1* i l'*usuari2*.

- **user_selector():** Crea un menú desplegable amb els usuaris que tinguin un nom semblant al passat com a paràmetre a l'URL.

- **populate_search():** Crea una llista d'usuaris amb el nom semblant al passat com a paràmetre a l'URL.

- **searchable_users(usuari,like):** Retorna una llista dels usuaris els quals el seu nom d'usuari s'assembla a *like*, i els quals no tenen cap relació d'amistat amb *usuari*.

6.2.1.3.3 Rànquing

Aquest mòdul controlador està lligat amb el mòdul *Rànquing* de les vistes.

Les funcions d'aquest controlador són les següents:

- **index():** Només uneix l'URL amb la corresponent vista.

- **main():** Proporciona a la vista *main* llistes ordenades de velocitat i distància recorreguda pels usuaris que són amics de l'usuari que ha cridat la funció i de tots els usuaris del servidor que no tenen la privacitat com ha privat.

- **km_ranquing_friends():** Retorna una llista no ordenada de la distància transcorreguda pels amics i l'usuari de qui ha cridat la funció.

- **km_ranquing_global():** Retorna una llista no ordenada de la distància transcorreguda per tots els usuaris.

- **speed_ranquing_friends():** Retorna una llista no ordenada de la velocitat mitjana realitzada pels amics i l'usuari de l'usuari que ha cridat la funció.

- **speed_ranquing_global():** Retorna una llista no ordenada de la velocitat mitjana realitzada per tots els usuaris.

- **getFriends(usuari):** Retorna en forma de diccionari els amics de l'usuari *usuari*.

- **getGlobal():** Retorna en forma de diccionari tots els usuaris que no tinguin privacitat privada.

- **getDistance(usuari):** Retorna la distància total recorreguda per les rutes de l'usuari *usuari*. Només té en compte les rutes les quals la seva privacitat és pública.

- **getSpeed(usuari):** Retorna la velocitat mitjana de les rutes de l'usuari *usuari*. Només té en compte les rutes on la seva privacitat és pública.

6.2.1.3.4 App_data

Aquest mòdul controlador és l'encarregat de gestionar la comunicació entre l'aplicació mòbil i el servei web. L'aplicació web mai inicia la comunicació, sempre respon a peticions. Les funcions d'aquest mòdul són les següents:

- **post_ruta():** Rep una ruta en el format *JSON* explicat a l'apartat 6.1.3.3.1, i insereix la ruta amb el seu corresponent *track* a la base de dades del servei web.

- **live_ruta():** Rep una ruta en el format *JSON* explicat a l'apartat 6.1.3.3.1, i insereix la ruta, si encara no ha estat inserida, i insereix el *track* rebut a la base de dades del servei web.

- **finish_live_ruta():** Rep la finalització d'una ruta en directa en el format *JSON* explicat a l'apartat 6.1.3.3.1.

- **update_pausa():** Rep l'avís d'aturada d'una ruta en directe en el format *JSON* explicat a l'apartat 6.1.3.3.1. Actualitza el camp pausa de l'últim *track* com a aturat.

- **get_rutes():** Retorna la llista de rutes amb el corresponent *track*, en el format *JSON* explicat a l'apartat 6.1.3.3.1, de l'usuari que ha fet la petició.

- **generateJsonTrack(sid_ruta):** Retorna el *track* de la ruta identificada per *sid_ruta*, en el format *JSON* explicat a l'apartat 6.1.3.3.1.

Finalment el mòdul *app_data* també proporciona l'*api rest*[3]. L'*api rest* és una interfície de l'aplicació que utilitza peticions HTTP per obtenir dades *GET*, *PUT*, *POST* i *DELETE*. L'*api rest* proporciona accés a les taules següents [19] a través de fer peticions a l'URL *https://mmas.pythonanywhere.com/app_data/api/*. Tot i que no totes tenen permisos per realitzar totes les peticions HTTP.

TAULA	GET	POST	PUT	DELETE
Rutes	✓	✓	✓	✓
Track	✓	✓	✓	✓
Amistat	✓	×	×	×

Figura 19: Taules Api rest

El format per obtenir la informació és el següent:

GET

- **/rutes[rutes]:** Fent una petició a *rutes/id_ruta* obtenim les dades d'aquesta ruta.
- **/rutes/rutes.nom_ruta.startswith:** Fent una petició seguint aquest patró obtenim la informació de les rutes que comencem pel text *rutes.nom_ruta.startswith*.
- **/rutes/rutes.nom_ruta/:field:** Fent una petició amb aquest patró obtenim el camp de la ruta o rutes amb *nom_ruta* igual a *rutes.nom_ruta*.
- **/track:** Fent una petició amb aquest patró obtenim tots els *tracks* del nostre usuari.
- **/track/rutes.nom_ruta:** Fent una petició amb aquest patró obtenim tots els *tracks* de les rutes amb *nom_ruta* igual a *rutes.nom_ruta*.
- **/track/rutes.nom_ruta/:field:** Fent una petició amb aquest patró obtenim el camp *field* dels *tracks* de les rutes amb *nom_ruta* igual a *rutes.nom_ruta*.
- **/amistat:** Fent una petició amb aquest patró obtenim totes les relacions d'amistat del nostre usuari.
- **/amistat/user.auth_username:** Fent una petició amb aquest patró obtenim totes les relacions d'amistat del nostre usuari on *id_usuari_1* o *id_usuari_2* sigui igual a la *id* de l'usuari *user.auth_username*.
- **/amistat/user.auth_username/:field:** Fent una petició amb aquest patró obtenim el camp *field* de totes les relacions d'amistat del nostre usuari on *id_usuari_1* o *id_usuari_2* sigui igual a la *id* de l'usuari *user.auth_username*.

POST

Per inserir una ruta o un *track* s'ha de passar la informació necessària en format *JSON* en la petició *POST*.

- Ruta:

```
{
    'id_ruta': '',
    'nom_ruta': '',
    'visibility': '',
    'live': '',
    'data': '',
    'total_time': '',
    'total_track': '',
    'total_dist': ''
}
```

- Track:

```
{
    'id_ruta': '',
    'latitude': '',
    'longitude': '',
    'timestamp': '',
    'pausa': ''
}
```

PUT

Per actualitzar una ruta o un *track* s'ha de fer una petició HTTP *PUT* a l'URL corresponent, */rutes/id_ruta* per les rutes o */track/id_track* pels *tracks*, i passar en format *JSON*, com en la petició *POST*, les dades a actualitzar. S'ha de tenir en compte que ni la *id_ruta* ni la *id_usuari* es poden modificar.

DELETE

Per esborrar una ruta o un *track* s'ha de fer una petició HTTP *DELETE* a l'URL corresponent, */rutes/id_ruta* per les rutes o */track/id_track* pels tracks. D'aquesta manera si ets propietari de la ruta o el *track* les dades s'esborraran.

7 Conclusions

En aconseguir desenvolupar una aplicació mòbil que permet obtenir, processar i gestionar les dades en línia o fora de línia sense la necessitat dels serveis de Google, podem dir que hem assolit el primer objectiu.

L'aplicació també proporciona la possibilitat de compartir rutes en directe utilitzant el servei web. Les rutes es comparteixen directament a la pàgina web, sense necessitat de cap tercer, d'aquesta manera s'assegura que les dades es mantinguin privades i sota el control de l'usuari.

Per tal d'assolir l'objectiu de proporcionar seguretat en les dades que transmetem, la pàgina web utilitza el protocol HTTPS.

S'ha proporcionat un control total als usuaris sobre les seves dades permetent modificar la privacitat d'aquestes i, fins i tot, esborrar-les totalment.

Finalment podem concloure que s'ha assolit l'objectiu principal de proporcionar una eina alternativa a Google que permetés obtenir, processar i gestionar les dades GPS, a més a més, demostrar que el control sobre les pròpies dades és una necessitat obligatòria no una funcionalitat extra.

Referències

[1] *Activity*. URL: https://developer.android.com/reference/android/app/Activity.

[2] *Android Studio*. URL: https://developer.android.com/studio/intro/.

[3] *Api rest*. URL: http://www.restapitutorial.com/.

[4] *AsyncTask*. URL: https://developer.android.com/reference/android/os/AsyncTask.

[5] *Context*. URL: https://developer.android.com/reference/android/content/Context.

[6] *Cursor*. URL: https://developer.android.com/reference/android/database/Cursor.

[7] *Django*. URL: https://www.djangoproject.com/.

[8] *Hypertrack*. URL: https://www.hypertrack.com/.

[9] *Intent*. URL: https://developer.android.com/reference/android/content/Intent.

[10] *Introducing JSON*. URL: https://www.json.org/.

[11] *Layout*. URL: https://developer.android.com/guide/topics/ui/declaring-layout.

[12] *Model-vista-presentador*. URL: https://ca.wikipedia.org/wiki/Model-vista-presentador.

[13] *Osmtracker*. URL: https://github.com/labexp/osmtracker-android/wiki.

[14] *Owntracks*. URL: http://owntracks.org/.

[15] *SharedPreferences*. URL: https://developer.android.com/reference/android/content/SharedPreferences.

[16] *Volley*. URL: https://developer.android.com/training/volley/.

[17] *Web2py*. URL: http://www.web2py.com/.

[18] *Xamarin*. URL: https://www.xamarin.com/.